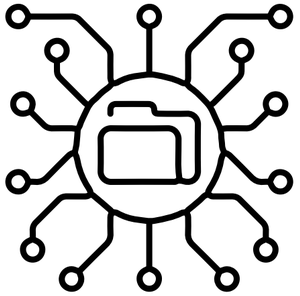
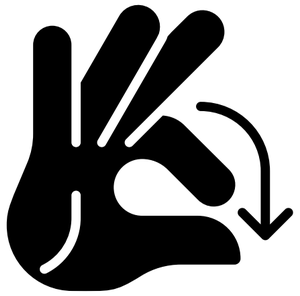
HARNESSING THE POWER OF DJANGO STREAMINGHTTPRESPONSE FOR EFFICIENT WEB STREAMING

“SHIVAM CHAURASIA”
SOFTWARE ARCHITECT
EPAM SYSTEMS

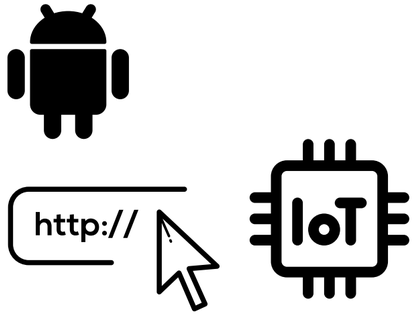
WHAT IS STREAMING DATA?



CONTINUOUSLY GENERATED



TYPICALLY IN SMALL SIZE



BY ANY SOURCE

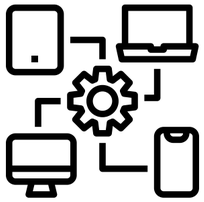
GIVE SOME OF EXAMPL E ?

DATA LOGS

IN GAME ACTIVITES

GEOSPATIAL SERVICES

TRADING PLATFORMS



CONNECTED DEVICES

DATA CENTER

SOCIAL MEDIA FEEDS

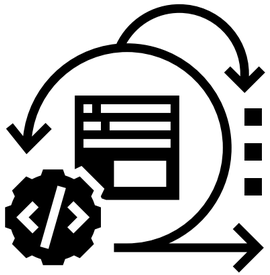
BENEFITS OF STREAMING DATA

REPORTING

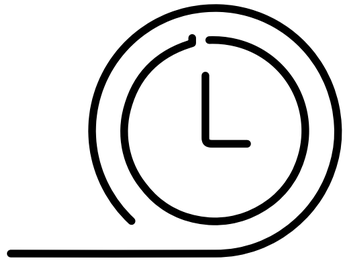
EMITING ALARMS

DEEPER INSIGHTS

HOW TO PROCESS STREAMINGDATA ?



INCREMENTALLY
RECORD-BY-RECORD



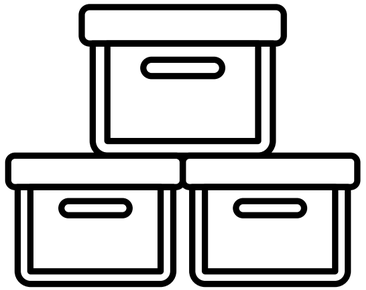
OVER SLIDING
TIME WINDOW

BUT BEFORE DEALING WITH STREAM DATA !

STREAM DATA	BATCH DATA
DATA SCOPE	
QUERIES ROLLING TIME WINDOW, OR JUST THE MOST RECENT DATA RECORD.	QUERIES OVER ALL OR MOST OF THE DATA IN THE DATASET.
DATA SIZE	
INDIVIDUAL RECORDS OR MICRO BATCHES OF A FEW RECORDS.	LARGE BATCHES OF DATA.
PERFORMANCE	
LATENCY OF SECONDS OR MILLISEC.	LATENCIES IN MINUTES TO HOURS.
ANALYSIS	
RESPONSE FUNCTIONS, AGGREGATES, AND METRICS.	COMPLEX ANALYTICS.

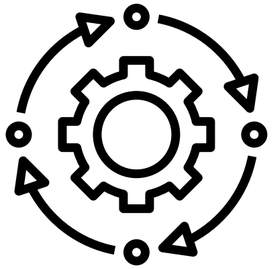
ARE THERE ANY CHALLENGES ?

LAYER 1 - STORAGE



STRONG CONSISTENCY
FAST, COST INEXPENSIVE,
REPLAYABLE READS AND
WRITES

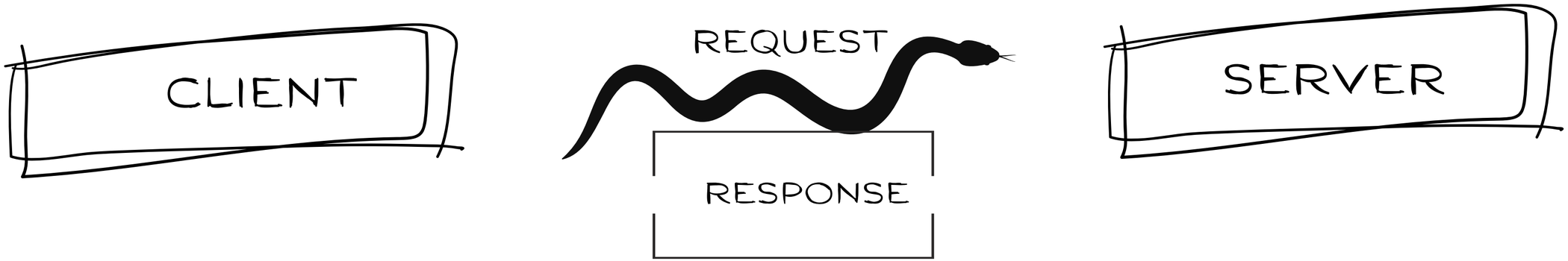
LAYER 2 - PROCESSING



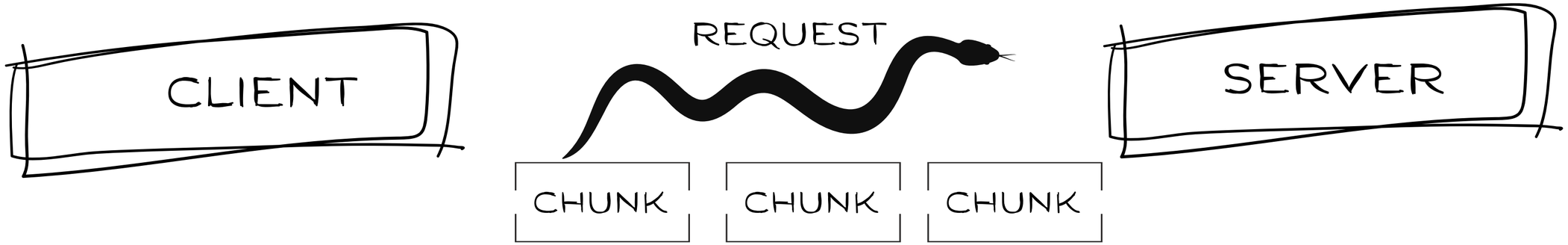
RUNNING COMPUTATIONS ,
DELETING DATA THAT IS NO
LONGER NEEDED.

SCALABILITY
DATA DURABILITY
FAULT TOLERANCE

DJANGO HTTP RESPONSE



DJANGOSTREAMING HTTP RESPONSE



WHAT'S THE DIFFERENCE ?

HTTPRESPONSE	STREAMING HTTPRESPONSE
ENITIRE RESPONSE BUILD IN MEMORY	NO NEED TO LOAD THE ENTIRE DATA IN MEMORY.
DISPATCHES IN SINGLE OBJECT	MULTIPLE CHUNKS
SHORT LIVED	LONG LIVED

WHAT'S THE DIFFERENCE ?

HTTPRESPONSE

```python
def pycon(request):
    message = 'Welcome To PyCon'
    response =  HttpResponse(message)
    response['Content-Length'] =
len(message)
    return response
```

STREAMING HTTPRESPONSE

```python
def pycon():
    yield 'Hello,'
    yield 'Welcome to PyCon!'

def my_view(request):
    # NOTE: No Content-Length header!
    return
StreamingHttpResponse(pycon)
```

STREAMING HTTPRESPONSE

TAKES ITERATOR WHICH YIELDS BYTESTRING OR STRING

DEPENDING ON GATEWAY INTERFACE(WSGI/ASGI) ITERATOR CAN BE SYNC OR ASYNC

IT DOESN'T RETURN CONTENT-LENGTH HEADER

TRANSFER ENCODING ?


```
❯ curl -vv --raw "http://192.168.99.100/csv_stream"
*   Trying 192.168.99.100...
* Connected to 192.168.99.100 (192.168.99.100) port 80
> GET /download_csv_streaming HTTP/1.1
> Host: 192.168.99.100
> User-Agent: curl/7.43.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: nginx/1.11.1
< Date: Fri, 29 Jul 2016 14:27:58 GMT
< Content-Type: text/csv
< Transfer-Encoding: chunked  <<<<< Chunked  Transfer
< Connection: keep-alive
< X-Frame-Options: SAMEORIGIN
```

EVENT STREAM FORMATS

CONTENT TYPE : TEXT/EVENT-STREAMS
BODY: DATA: LINE, FOLLOWED BY YOUR MESSAGE

DATA: MY MESSAGE\N\N

FOLLOWED BY TWO "\N" CHARACTERS TO END THE STREAM

EVENT STREAM PROCESSING


```python
def stream_csv(request):
    def event_stream():
        with open('stream_source', 'r') as csv_file:
            csv_reader = csv.reader(csv_file)
            for row in csv_reader:
                yield f'data: {row}\n\n'

    response = StreamingHttpResponse(
        event_stream(),
        content_type='text/event-stream')
    response['Cache-Control'] = 'no-cache'
    response['Content-Type'] = 'text/event-stream'
    return response
```

EVENT STREAM PROCESSING

Web Inspector — 127.0.0.1 — dashboard

Elements | Console | Sources | Network | Timelines | Storage | Graphics

Filter Full URL | Other

Name	Domain	Type	Transfer Size	Time
csv_stream	127.0.0.1	eventsource	465 B	10.3ms
csv_stream	127.0.0.1	eventsource	465 B	3.89ms
csv_stream	127.0.0.1	eventsource	465 B	8.79ms
csv_stream	127.0.0.1	eventsource	465 B	11.1ms
csv_stream	127.0.0.1	eventsource	465 B	11.0ms
csv_stream	127.0.0.1	eventsource	465 B	11.4ms
csv_stream	127.0.0.1	eventsource	465 B	9.57ms
csv_stream	127.0.0.1	eventsource	465 B	5.59ms
csv_stream	127.0.0.1	eventsource	465 B	5.37ms
csv_stream	127.0.0.1	eventsource	465 B	4.74ms

5.00s 10.00s

STOCK MARKET UPDATE

```python
from django.http import StreamingHttpResponse
from channels.layers import get_channel_layer
from asgiref.sync import async_to_sync
import json
import time
def generate_stock_data():
    # Connect to a real-time stock data source (e.g., WebSocket, API)
    # Iterate and yield stock data in chunks
    while True:
        stock_data = get_real_time_stock_data()
        yield json.dumps(stock_data)
        # Simulate delay between data updates
        time.sleep(1)

def stream_stock_data(request):
    channel_layer = get_channel_layer()
    stream_name = "stock_data_stream"

    def send_stock_data():
        for stock_data in generate_stock_data():
            async_to_sync(channel_layer.group_send)(stream_name, {
                "type": "stock_data_update",
                "data": stock_data
            })
            # Yield the data for streaming response as well
            yield stock_data
    response = StreamingHttpResponse(send_stock_data(), content_type="application/json")
    # Open a channel to stream data to clients via WebSocket or other streaming
protasync_to_sync(channel_layer.group_add)(stream_name, request.channel_name)
    def handle_disconnect():
        # Cleanup and close the channel when the client disconnects
        async_to_sync(channel_layer.group_discard)(stream_name, request.channel_name)
    response.streaming_content.disconnect = handle_disconnect
    return response
```

STREAMINGHTTPRESPONSE ATTRIBUTES

.STREAMING_CONTENT - ITERATOR OF RESPONSE

.IS_ASYNC - BOOLEAN

.STATUS_CODE - HTTP STATUS CODES

.STREAMING - ALWAYS TRUE

BEST PRACTICES

CHUNK SIZE

DATA COMPRESSION

WEB SERVER COMPATIBILITY